iSpike 2.1 Manual

1. Introduction

iSpike is a C++ library that interfaces between spiking neural network simulators and the iCub robot. It uses a biologically-inspired approach to convert the robot's sensory information into spikes that are passed to the neural network simulator, and it decodes output spikes from the network into motor signals that are sent to control the robot. Applications of iSpike range from embodied models of the brain to the development of intelligent robots using biologically-inspired spiking neural networks. iSpike is an open source library that is available for free download under the terms of the GPL.

A full description of iSpike is available in the paper LazdinsFidjelandGamez11_iSpike.pdf, which is included in the doc folder of the release, which also includes source code documentation. This manual gives instructions on how to build iSpike and use it with your application.

2. Installing iSpike

2.1 Windows

iSpike is available as a binary release for Windows consisting of a dynamic library, libiSpike.dll, include files and the third party dependencies. The Windows release also includes mingw32 builds of the Boost libraries regex, system and thread, which are needed to run the library. These libraries (in the 'thirdparty' folder) need to be on the system path for iSpike to work correctly. The easiest way to do this is to place them in the same directory as iSpike.

The source code of iSpike is included in the release or you can download the latest version from http://sourceforge.net/projects/ispike/develop. Note that the latest development version may not be as stable as the version included with the release.

To build iSpike from source you will need Boost and CMake. If you are building using MinGW you can link against the Boost libraries included in the thirdparty folder and download the header files for Boost 1.46. If you are using Visual Studio, download the latest libraries and header files for Boost. Before building, set BOOST_ROOT in src/CMakeLists.txt to the root directory of your Boost installation.

2.2 Linux

The source code of iSpike is included in the release or you can download the latest version from http://sourceforge.net/projects/ispike/develop. Note that the latest development version may not be as stable as the version included with the release.

iSpike depends on Boost and you will need CMake to configure the build. Before building set BOOST_ROOT in src/CMakeLists.txt to the root directory of your Boost installation. This line can be commented out if Boost is installed in a standard library location.

When you have configured the build follow the same build steps as for Mac OS X.

2.3 Mac OS X

A binary release of iSpike is available as a dmg file. Double click it to mount it as a drive and then double click on the .pkg file to install the libraries. iSpike will need to have the Boost libraries regex, system and thread on the system path. These are installed in the same directory as the iSpike library.

To build iSpike from source you will need Boost, which can be obtained from http://www.macports.org/. Install Boost using the command “sudo port install boost”. You will also need to install cmake from http://www.cmake.org. The build steps are as follows.

1. Download the latest source code from the iSpike website.
2. Change to the root of the source code and create a directory called ‘build’.
3. Change to the build directory and type ‘cmake-gui ..’
4. Press “Configure” and “Generate” to create the Makefile.
5. Return to the command line and type ‘make’ followed by ‘make install’
6. Consult the cmake documentation if you have problems during this process.

3. Using iSpike

iSpike consists of the following components:

- *Reader*. Extracts sensory data of a given type from a given location. The current release of iSpike has a FileAngleReader that reads an angle from a file, a FileVisualReader that reads a .ppm image from a file, a YarpAngleReader that reads joint angles from YARP and a YarpVisualReader that reads images from YARP. Readers run as separate threads that buffer the latest data.
- *Writer*. Outputs data of a given type to a given destination. The current release has a FileAngleWriter that writes joint angles to a file as well as a YarpAngleWriter that sends motor commands to the iCub using YARP.
- *Input Channel*. Receives sensory data from a Reader of a given type and transforms it into a spike representation, which is passed to the neural simulator. The current release has a JointInputChannel that converts the angular data prepared by an AngleReader into spikes and a VisualInputChannel that converts visual data prepared by a VisualReader into spikes. The spike conversion process depends on simulated Izhikevich neurons, which are stepped in synchrony with the neural network simulator.
- *Output Channel*. Receives a spike pattern from the neural simulator, converts it into an appropriate format, and uses a Writer to deliver it to a predefined destination. The current release has a JointOutputChannel that converts neuron spike patterns into joint angle motor commands and uses an AngleWriter to deliver these to the iCub. The spike conversion process depends on simulated Izhikevich neurons, which are stepped in synchrony with the neural network simulator.

To use iSpike to convert real valued data into spikes, you start by creating a reader, which interacts with YARP or reads data from a file. For example, the following code creates a reader that loads an image from the file `inputImage.ppm':

```
// Create reader
FileVisualReader* visualReader = new FileVisualReader();

// Set the properties of the reader (in this case the name of the file to read in)
map<string, Property> readerProperties = visualReader->getProperties();
readerProperties.at("File Name") = Property("inputImage.ppm", "File Name", "description", true);
visualReader->initialize(readerProperties);
```

The next stage is the creation of a visual input channel that converts the image provided by the reader into spikes:

```
// Create visual input channel
VisualInputChannel* visualChannel = new VisualInputChannel();
```

```
// Set the properties of the channel (in this case the radius of the fovea)
map<string, Property> visualChannelProperties = visualChannel->getProperties();
visualChannelProperties.at("Fovea Radius") = Property(Property::Double,
              50, "Fovea Radius", "description", true);

// Initialize channel with the properties and reader.
visualChannel->initialize(visualReader, visualChannelProperties);
```

The spike conversion process depends on simulated Izhikevich neurons, which are stepped with a temporal resolution of 1~ms per time step in synchrony with the neural network simulator. At each time step the spikes resulting from the conversion of the visual image are extracted and passed to the neural simulator:

```
// Step through 1 second of simulation
for(int i=0; i<1000; ++i){
       // Extract spikes from visual channel
       visualChannel->step();
       const vector<unsigned>& firingNeuronIDs = visualChannel->getFiring();

       // Pass spikes to neural simulator
       ...
}
```

The construction of writers and output channels works in a similar way to the construction of readers and input channels. The key difference is that spikes from the neural simulator are passed into iSpike at each time step:

```
// Step through 1 second of simulation
for(int i=0; i<1000; ++i){
       // Extract spikes from neural simulator
       const vector<unsigned>& firingNeuronIDs = ...

       // Pass spikes to iSpike output channel
       motorOutputChannel->setFiring(firingNeuronIDs);
}
```

4. iSpike and SpikeStream

A graphical interface for iSpike has been created within the SpikeStream simulation and analysis environment, which enables the user to configure iSpike and connect it to networks simulated within SpikeStream. More information is available in the SpikeStream manual, which is available at: http://spikestream.sourceforge.net/pages/documentation.html

5. Questions & Problems

Support for iSpike is available at: ispike-user@lists.sourceforge.net.